

ATTORNEY'S DOCKET NO. 06502.0367
SUN REF. NO. P5219
PATENTS

VIA EXPRESS MAIL - EL396216463US

UNITED STATES PATENT APPLICATION

OF

Guy L. STEELE Jr.

FOR

**FLOATING POINT SQUARE ROOT PROVIDER WITH EMBEDDED STATUS
INFORMATION**

10636007

RELATED APPLICATIONS

[001] U.S. Patent Application Serial No. _____, filed on even date herewith in the name of Guy L. Steele Jr. and entitled "Floating Point System That Represents Status Flag Information Within a Floating Point Operand," assigned to the assignee of the present application, is hereby incorporated by reference.

DESCRIPTION OF THE INVENTION

Field of the Invention

[002] The invention relates generally to systems and methods for performing floating point operations, and more particularly to systems and methods for performing a floating point square root with embedded status information associated with a floating point operand.

Background of the Invention

[003] Digital electronic devices, such as digital computers, calculators and other devices, perform arithmetic calculations on values in integer, or "fixed point," format, in fractional, or "floating point" format, or both. Institute of Electrical and Electronic Engineers (IEEE) Standard 754, (hereinafter "IEEE Std. 754" or "the Standard") published in 1985 and adopted by the American National Standards Institute (ANSI), defines several standard formats for expressing values in floating point format and a number of aspects regarding behavior of computation in connection therewith. In accordance with IEEE Std. 754, a representation in floating point format comprises a plurality of binary digits, or "bits," having the structure

[004]
$$se_{msb} \cdots e_{lsb} f_{msb} \cdots f_{lsb}$$

where bit “s” is a sign bit indicating whether the entire value is positive or negative, bits “ $e_{msb} \cdots e_{lsb}$ ” comprise an exponent field represent the exponent “e” in unsigned binary biased format, and bits “ $f_{msb} \cdots f_{lsb}$ ” comprise a fraction field that represents the fractional portion “f” in unsigned binary format (“msb” represents “most significant bit” and “lsb” represents “least significant bit”). The Standard defines two general formats. A “single” format comprises thirty-two bits while a “double” format comprises sixty-four bits. In the single format, there is one sign bit “s,” eight bits “ $e_7 \dots e_0$ ” comprising the exponent field and twenty-three bits “ $f_{22} \dots f_0$ ” comprising the fraction field. In the double format, there is one sign bit “s,” eleven bits “ $e_{10} \dots e_0$ ” comprising the exponent field and fifty-two bits “ $f_{51} \dots f_0$ ” comprising the fraction field.

[005] As indicated above, the exponent field of the floating point representation “ $e_{msb} \cdots e_{lsb}$ ” represents the exponent “E” in biased format. The biased format provides a mechanism by which the sign of the exponent is implicitly indicated. In particular, the bits “ $e_{msb} \cdots e_{lsb}$ ” represent a binary encoded value “e” such that “ $e = E + \text{bias}$.” This allows the exponent E to extend from -126 to +127, in the eight-bit “single” format, and from -1022 to +1023 in the eleven-bit “double” format, and provides for relatively easy manipulation of the exponents in multiplication and division operations, in which the exponents are added and subtracted, respectively.

[006] IEEE Std. 754 provides for several different formats with both the single and double formats which are generally based on the bit patterns of the bits “ $e_{msb} \cdots e_{lsb}$ ” comprising the exponent field and the bits “ $f_{msb} \cdots f_{lsb}$ ” comprising the fraction field. If a number is represented such that all of the bits “ $e_{msb} \cdots e_{lsb}$ ” of the

exponent field are binary one's (*i.e.*, if the bits represent a binary-encoded value of "255" in the single format or "2047" in the double format) and all of the bits " $f_{msb} \cdots f_{lsb}$ " of the fraction field are binary zeros, then the value of the number is positive or negative infinity, depending on the value of the sign bit "s." In particular, the value "v" is $v = (-1)^s \infty$, where " ∞ " represents the value "infinity." On the other hand, if all of the bits " $e_{msb} \cdots e_{lsb}$ " of the exponent field are binary one's and if the bits " $f_{msb} \cdots f_{lsb}$ " of the fraction field are not all zero's, then the value that is represented is deemed "not a number," which is abbreviated in the Standard by "NaN."

[007] If a number has an exponent field in which the bits " $e_{msb} \cdots e_{lsb}$ " are neither all binary ones nor all binary zeros (*i.e.*, if the bits represent a binary-encoded value between 1 and 254 in the single format or between 1 and 2046 in the double format), the number is said to be in a "normalized" format. For a number in the normalized format, the value represented by the number is

$$v = (-1)^s 2^{e - \text{bias}} (1. |f_{msb} \cdots f_{lsb}|), \text{ where } "|" \text{ represents a concatenation operation.}$$

Effectively, in the normalized format, there is an implicit most significant digit having the value "one," so that the twenty-three digits in the fraction field of the single format, or the fifty-two digits in the fraction field of the double format, will effectively represent a value having twenty-four digits or fifty-three digits of precision, respectively, where the value is less than two, but not less than one.

[008] On the other hand, if a number has an exponent field in which the bits " $e_{msb} \cdots e_{lsb}$ " are all binary zeros, representing the binary-encoded value of "zero," and a fraction field in which the bits " $f_{msb} \cdots f_{lsb}$ " are not all zero, the number is said to be in a "de-normalized" format. For a number in the de-normalized format, the value

represented by the number is $v = (-1)^s 2^{e-bias+1} (0.f_{msb} \cdots f_{lsb})$. It will be appreciated that the range of values of numbers that can be expressed in the de-normalized format is disjoint from the range of values of numbers that can be expressed in the normalized format, for both the single and double formats. Finally, if a number has an exponent field in which the bits " $e_{msb} \cdots e_{lsb}$ " are all binary zeros, representing the binary-encoded value of "zero," and a fraction field in which the bits $f_{msb} \cdots f_{lsb}$ are all zero, the number has the value "zero". It will be appreciated that the value "zero" may be positive zero or negative zero, depending on the value of the sign bit.

[009] Generally, circuits or devices that perform floating point computations or operations (generally referred to as floating point units) conforming to IEEE Std. 754 are designed to generate a result in three steps:

[010] (a) In the first step, an approximation calculation step, an approximation to the absolutely accurate mathematical result (assuming that the input operands represent the specific mathematical values as described by IEEE Std. 754) is calculated that is sufficiently precise as to allow this accurate mathematical result to be summarized. The summarized result is usually represented by a sign bit, an exponent (typically represented using more bits than are used for an exponent in the standard floating-point format), and some number "N" of bits of the presumed result fraction, plus a guard bit and a sticky bit. The value of the exponent will be such that the value of the fraction generated in step (a) consists of a 1 before the binary point and a fraction after the binary point. The bits are commonly calculated so as to obtain the same result as the following conceptual procedure (which is impossible under some circumstances to carry out in practice):

calculate the mathematical result to an infinite number of bits of precision in binary scientific notation, and in such a way that there is no bit position in the significand such that all bits of lesser significance are 1-bits (this restriction avoids the ambiguity between, for example, 1.100000... and 1.011111... as representations of the value "one-and-one-half"); let the N most significant bits of the infinite significand be used as the intermediate result significand; let the next bit of the infinite significand be the guard bit; and let the sticky bit be 0 if and only if ALL remaining bits of the infinite significand are 0-bits (in other words, the sticky bit is the logical OR of all remaining bits of the infinite fraction after the guard bit).

[011] (b) In the second step, a rounding step, the guard bit, the sticky bit, perhaps the sign bit, and perhaps some of the bits of the presumed significand generated in step (a) are used to decide whether to alter the result of step (a). For conventional rounding modes defined by IEEE Std. 754, this is a decision as to whether to increase the magnitude of the number represented by the presumed exponent and fraction generated in step (a). Increasing the magnitude of the number is done by adding 1 to the significand in its least significant bit position, as if the significand were a binary integer. It will be appreciated that, if the significand is all 1-bits, then the magnitude of the number is "increased" by changing it to a high-order 1-bit followed by all 0-bits and adding 1 to the exponent.

[012] Regarding the rounding modes, it will be further appreciated that,

[013] (i) if the result is a positive number, and

[014] (a) if the decision is made to increase, effectively the decision has been made to increase the value of the result, thereby rounding the result up (*i.e.*, towards positive infinity), but

[015] (b) if the decision is made not to increase, effectively the decision has been made to decrease the value of the result, thereby rounding the result down (*i.e.*, towards negative infinity); and

[016] (ii) if the result is a negative number, and

[017] (a) if the decision is made to increase, effectively the decision has been made to decrease the value of the result, thereby rounding the result down, but

[018] (b) if the decision is made not to increase, effectively the decision has been made to increase the value of the result, thereby rounding the result up.

[019] (c) In the third step, a packaging step, a result is packaged into a standard floating-point format. This may involve substituting a special representation, such as the representation defined for infinity or NaN if an exceptional situation (such as overflow, underflow, or an invalid operation) was detected. Alternatively, this may involve removing the leading 1-bit (if any) of the fraction, because such leading 1-bits are implicit in the standard format. As another alternative, this may involve shifting the fraction in order to construct a denormalized number. As a specific example, it is assumed that this is the step that forces the result to be a NaN if any input operand is a NaN. In this step, the decision is also made as to whether the result should be an infinity. It will be appreciated that, if the

result is to be a NaN or infinity from step (b), the original result will be discarded and an appropriate representation will be provided as the result.

[020] In addition in the packaging step, floating point status information is generated, which is stored in a floating point status register. The floating point status information generated for a particular floating point operation includes indications, for example, as to whether

[021] (i) a particular operand is invalid for the operation to be performed (“invalid operation”);

[022] (ii) if the operation to be performed is division, the divisor is zero (“division-by-zero”);

[023] (iii) an overflow occurred during the operation (“overflow”);

[024] (iv) an underflow occurred during the operation (“underflow”);

and

[025] (v) the rounded result of the operation is not exact (“inexact”).

[026] These conditions are typically represented by flags that are stored in the floating point status register. The floating point status information can be used to dynamically control the operations in response to certain instructions, such as conditional branch, conditional move, and conditional trap instructions that may be in the instruction stream subsequent to the floating point instruction. Also, the floating point status information may enable processing of a trap sequence, which will interrupt the normal flow of program execution. In addition, the floating point status information may be used to affect certain ones of the functional unit control signals

that control the rounding mode. IEEE Std. 754 also provides for accumulating floating point status information from, for example, results generated for a series or plurality of floating point operations.

[027] IEEE Std. 754 has brought relative harmony and stability to floating-point computation and architectural design of floating-point units. Moreover, its design was based on some important principles, and rests on a sensible mathematical semantics that eases the job of programmers and numerical analysts. It also supports the implementation of interval arithmetic, which may prove to be preferable to simple scalar arithmetic for many tasks. Nevertheless, IEEE Std. 754 has some serious drawbacks, including:

[028] (i) Modes (*e.g.*, the rounding modes and traps enabled/disabled mode), flags (*e.g.*, flags representing the status information), and traps required to implement IEEE Std. 754 introduce implicit serialization issues. Implicit serialization is essentially the need for serial control of access (read/write) to and from globally used registers, such as a floating point status register. Under IEEE Std. 754, implicit serialization may arise between (1) different concurrent floating-point instructions and (2) between floating point instructions and the instructions that read and write the flags and modes. Furthermore, rounding modes may introduce implicit serialization because they are typically indicated as global state, although in some microprocessor architectures, the rounding mode is encoded as part of the instruction operation code, which will alleviate this problem to that extent. Thus, the potential for implicit serialization makes the Standard difficult to

implement coherently and efficiently in today's superscalar and parallel processing architectures without loss of performance.

[029] (ii) The implicit side effects of a procedure that can change the flags or modes can make it very difficult for compilers to perform optimizations on floating point code. As a result, compilers for most languages usually assume that every procedure call is an optimization barrier in order to be safe. This unfortunately may lead to further loss of performance.

[030] (iii) Global flags, such as those that signal certain modes, make it more difficult to do instruction scheduling where the best performance is provided by interleaving instructions of unrelated computations. Thus, instructions from regions of code governed by different flag settings or different flag detection requirements cannot easily be interleaved when they must share a single set of global flag bits.

[031] (iv) Furthermore, traps have been difficult to integrate efficiently into computing architectures and programming language designs for fine-grained control of algorithmic behavior.

[032] Thus, there is a need for a system that avoids such problems when performing floating point operations and, in particular, when performing a floating point square root with embedded status information associated with a floating point operand.

SUMMARY OF THE INVENTION

[033] Consistent with the current invention, a floating point square root provider with embedded status information method and system are provided that

avoid the problems associated with prior art floating point square root provider systems as discussed herein above.

[034] In one aspect, a system for providing a floating point square root comprises an analyzer circuit configured to determine a first status of a first floating point operand based upon data within the first floating point operand. In addition, the system comprises a results circuit coupled to the analyzer circuit and configured to assert a resulting floating point operand containing the square root of the first floating point operand and a resulting status embedded with the resulting floating point operand.

[035] In another aspect, a method for providing a floating point square root comprises determining a first status of a first floating point operand based upon data within the first floating point operand. In addition, the method comprises asserting a resulting floating point operand containing the square root of the first floating point operand and a resulting status embedded with the resulting floating point operand.

[036] In yet another aspect, a computer-readable medium on which is stored a set of instructions for providing a floating point square root, which when executed perform stages comprising determining a first status of a first floating point operand based upon data within the first floating point operand. The instruction set further comprises asserting a resulting floating point operand containing the square root of the first floating point operand and a resulting status embedded with the resulting floating point operand.

[037] Both the foregoing general description and the following detailed description are exemplary and are intended to provide further explanation of the invention as claimed.

BRIEF DESCRIPTION OF THE DRAWINGS

[038] The accompanying drawings provide a further understanding of the invention and, together with the detailed description, explain the principles of the invention. In the drawings:

[039] FIG. 1 is a functional block diagram of an exemplary system for providing a floating point square root consistent with an embodiment of the present invention;

[040] FIG. 2 illustrates exemplary formats for representations of floating point values generated by the system of FIG. 1 consistent with an embodiment of the present invention;

[041] FIG. 3 illustrates a table useful in understanding the operations of the exemplary system of FIG. 1 consistent with an embodiment of the present invention; and

[042] FIG. 4 depicts exemplary patterns of input and output signals received and generated by a square root decision table logic circuit used in the exemplary system of FIG. 1 consistent with an embodiment of the present invention.

DESCRIPTION OF THE EMBODIMENTS

[043] Reference will now be made to various embodiments according to this invention, examples of which are shown in the accompanying drawings and will be obvious from the description of the invention. In the drawings, the same reference

numbers represent the same or similar elements in the different drawings whenever possible.

[044] Related U.S. Patent Application Serial No. _____, which has previously been incorporated by reference, describes an exemplary floating point unit in which floating point status information is encoded in the representations of the results generated thereby. The exemplary floating point unit includes a plurality of functional units, including an adder unit, a multiplier unit, a divider unit, a square root unit, a maximum/minimum unit, a comparator unit and a tester unit, all of which operate under control of functional unit control signals provided by a control unit. The present application is directed to an exemplary square root unit that can be used in floating point operations with the floating point unit described in related U.S. Patent Application Serial No. _____.

[045] FIG. 1 is a functional block diagram of an exemplary square root unit 10 constructed in accordance with an embodiment of the invention. Generally, the square root unit 10 receives one floating point operand and generates therefrom a result and, in some cases, floating point status information, with the floating point status information being encoded in and comprising part of the floating point representation of the result. Since the floating point status information comprises part of the floating point representation of the result, instead of being separate and apart from the result as in prior art square root units, the implicit serialization that is required by maintaining the floating point status information separate and apart from the result can be obviated.

[046] The square root unit 10 encodes the floating point status information in results that are generated in certain formats. This will be illustrated in connection with FIG. 2. FIG. 2 depicts exemplary formats of floating point operands that the square root unit 10 may receive and of results that it generate. With reference to the embodiment illustrated in FIG. 2, seven formats are depicted, including a zero format 70, an underflow format 71, a denormalized format 72, a normalized non-zero format 73, an overflow format 74, an infinity format 75 and a not-a-number (NaN) format 76. The zero format 70 is used to represent the values "zero," or, more specifically, positive or negative zero, depending on the value of "s," the sign bit.

[047] The underflow format 71 provides a mechanism by which the square root unit 10 can indicate that the result of a computation is an underflow. In the underflow format, the sign bit "s" indicates whether the result is positive or negative, the bits $e_{msb} \dots e_{lsb}$ of the exponent field are all binary zero's, and the bits $f_{msb} \dots f_{lsb+1}$ of the fraction field, except for the least significant bit, are all binary zero's. The least significant bit f_{lsb} of the fraction field is a binary one.

[048] The denormalized format 72 and normalized non-zero format 73 are used to represent finite non-zero floating point values substantially along the lines of that described above in connection with IEEE Std. 754. In both formats 72 and 73, the sign bit "s" indicates whether the result is positive or negative. The bits $e_{msb} \dots e_{lsb}$ of the exponent field of the denormalized format 72 are all binary zero's, whereas the bits $e_{msb} \dots e_{lsb}$ of the exponent field of the normalized non-zero format 73 are mixed one's and zero's, except that the exponent field of the normalized non-zero format 73 will not have a pattern in which bits $e_{msb} \dots e_{lsb+1}$, are all binary ones and the least

significant bit e_{lsb} zero and all of the bits $f_{msb}...f_{lsb}$ of the fraction field are all binary one's. In format 72, the bits $f_{msb}...f_{lsb}$ of the fraction field are not all binary zero's.

[049] The overflow format 74 provides a mechanism by which the square root unit 10 can indicate that the result of a computation is an overflow. In the overflow format 74, the sign bit "s" indicates whether the result is positive or negative, the bits $e_{msb} \dots e_{lsb+1}$ of the exponent field are all binary ones, with the least significant bit e_{lsb} being zero. The bits $f_{msb} \dots f_{lsb}$ of the fraction field are all binary ones.

[050] The infinity format 75 provides a mechanism by which the square root unit 10 can indicate that the result is infinite. In the infinity format 75, the sign bit "s" indicates whether the result is positive or negative, the bits $e_{msb}...e_{lsb}$ of the exponent field are all binary ones, and the bits $f_{msb}...f_{lsb+5}$ of the fraction field are all binary zero's. The five least significant bits $f_{lsb+4}...4_{lsb}$ of the fraction field are flags, which will be described below.

[051] The NaN format 76 provides a mechanism by which the square root unit can indicate that the result is not a number. In the NaN format, the sign bit "s" can be any value, the bits $e_{msb} \dots e_{lsb}$ of the exponent field are all binary ones, and the bits $f_{msb} \dots f_{lsb+5}$ of the fraction field are not all binary zero's. The five least significant bits $f_{lsb+4} \dots f_{lsb}$ of the fraction field are flags, which will be described below.

[052] As noted above, in values represented in the infinity format 75 and the NaN format 76, the five low order bits $f_{lsb+4} \dots f_{lsb}$ of the fraction field are flags. In the formats used with the square root unit 10, the five flags include the flags that are defined by IEEE Std. 754, including an invalid operation flag "n," an overflow flag "o," an underflow flag "u," a division-by-zero flag "z," and an inexact flag "x." For

example, a value in the NaN format 76 in which both the overflow flag "o" and the division-by-zero flag "z" are set indicates that the value represents a result of a computation that involved an overflow (this from the overflow flag "o"), as well as an attempt to divide by zero (this from the division-by-zero flag "z"). It should be noted that the flags provide the same status information as provided by, for example, information stored in a floating point status register in a prior art floating point unit that includes square root unit 10, but, since the information is provided as part of the result and stored therewith in registers in which the result is ultimately stored, multiple instructions can be contemporaneously executed. This is because the floating point status information that may be generated during execution of one instruction, when stored, will not over-write previously-stored floating point status information generated during execution of another instruction.

[053] In addition to including status information in the five low-order bits $f_{lsb+4} \dots f_{lsb}$ of the fraction field for values in the NaN format 76, other information can also be encoded in the next five low-order bits $f_{lsb+9} \dots f_{lsb+5}$. If the value in the NaN format 76 is the result of an operation, the other information indicates the operation and types of operands that gave rise to the result. In one embodiment, the other information is associated with binary encoded values (BEV) of those bits $f_{lsb+9} \dots f_{lsb+5}$ as follows:

Bit Pattern Of Result	BEV of $f_{lsb+9} \dots f_{lsb+5}$	Meaning
s 11111111 00000000000000010nouzx	0 or 1	no specific meaning
s 11111111 00000000000000011nouzx	2	infinity minus infinity
s 11111111 000000000000000100nouzx	3	OV minus OV
s 11111111 000000000000000101nouzx	4	zero times infinity
s 11111111 000000000000000101nouzx	5	UN times OV
s 11111111 00000000000001000nouzx	6 or 7	no specific meaning
s 11111111 00000000000001000nouzx	8	zero divided by zero

s 11111111 00000000000001001nouzx	9	infinity divided by infinity
s 11111111 00000000000001010nouzx	10	UN divided by UN
s 11111111 00000000000001011nouzx	11	OV divided by OV
s 11111111 00000000000001100nouzx	12	square root of less than zero
	13-16	no specific meaning
s 11111111 00000000000010001nouzx	17	remainder by zero
s 11111111 00000000000010010nouzx	18	remainder by UN
s 11111111 00000000000010011nouzx	19	remainder by OV
s 11111111 06000000000010100nouzx	20	remainder of infinity
s 11111111 00000000000010101nouzx	21	remainder of infinity by zero
s 11111111 00000000000010110nouzx	22	remainder of infinity by UN
s 11111111 00000000000010111nouzx	23	remainder of infinity by OV
s 11111111 00000000000011000nouzx	24	remainder of OV
s 11111111 00000000000011001nouzx	25	remainder of OV by zero
s 11111111 00000000000011010nouzx	26	remainder of OV by UN
s 11111111 00000000000011011nouzx	27	remainder of OV by OV
	28-31	no specific meaning

[054] In the following, it will be assumed that the formats represent thirty-two bit values; extension to, for example, sixty-four bit values or values represented in other numbers of bits will be readily apparent to those skilled in the art. Additionally, "OV" refers to an operand in the overflow format 74, "UN" refers to an operand in the underflow format 71 and "infinity" refers to an operand in the infinity format 75.

[055] With this background, the structure and operation of the square root unit 10 will be described in connection with FIG. 1. Before proceeding to that description, it will be convenient to have names for the finite nonzero numbers that are adjacent to +OV (a value in the overflow format 74 with the sign bit "s" having the value "zero" indicating a positive value), -OV (a value in the overflow format 74 with the sign bit "s" having the value "one," indicating a negative value), +UN (a value in the underflow format 71 with the sign bit "s" having the value "zero," indicating a positive value), and -UN (a value in the underflow format 71 with the sign bit "s" having the value "one," indicating a negative value), as follows:

[056] 0 00000000 0000000000000000000010 +TINY

[057] 1 00000000 000000000000000000000010 -TINY
 [058] 0 11111110 111111111111111111111110 +HUGE
 [059] 1 11111110 111111111111111111111110 -HUGE

[060] Generally, +OV can be deemed to refer to "some (or any) value that is strictly between +HUGE and $+\infty$ " and +UN can be deemed to refer to "some (or any) value that is strictly between +0 and +TINY". Similarly, -OV can be deemed to refer to "some (or any) value that is strictly between -HUGE and $-\infty$ " and -UN can be deemed to refer to "some (or any) value that is strictly between -0 and -TINY." These names will be used in the following description.

[061] With reference to FIG. 1, the exemplary square root unit 10 includes an operand buffer 11, an operand analysis circuit 12, a square root core 13, a result assembler 14, and a square root decision table logic circuit 15. The operand buffer 11 receives and stores an operand from, for example, a register (not shown) in a conventional manner. The square root core 13 receives the operand from the operand buffer 11, except as described below, and rounding mode information from, for example, a rounding mode store 16. The square root core 13 then generates a result in accordance with IEEE Std. 754. Square root core 13 is conventional and will not be described in detail herein.

[062] The operand analysis circuit 12 analyzes the operand in the operand buffer 11 and generates signals providing information relating thereto, which signals are provided to the square root decision table logic circuit 15. The result assembler 14 receives information from a number of sources, including the operand buffer 11, square root core 13 and several predetermined value stores as described below.

Under control of control signals from the square root decision table logic circuit 15, the result assembler assembles the result, which is provided on a result bus 17. The result bus 17, in turn, may deliver the result to any convenient destination, such as a register in a register set (not shown), for storage or other use.

[063] The system for providing a floating point square root may comprise an analyzer circuit configured to determine a first status of a first floating point operand based upon data within the first floating point operand. In one embodiment, the analyzer circuit includes buffer 11 and analysis circuit 12. In addition, the system for providing a floating point square root includes a results circuit coupled to the analyzer circuit. The results circuit is configured to assert a resulting floating point operand containing the square root of the first floating point operand and a resulting status embedded within the resulting floating point operand. The results circuit may be implemented with a square root circuit (comprising the square root core 13), the square root decision logic table circuit 15, and result assembler 14.

[064] Those skilled in the art will appreciate that the invention may be practiced in an electrical circuit comprising discrete electronic elements, packaged or integrated electronic chips containing logic gates, a circuit utilizing a microprocessor, or on a single chip containing electronic elements or microprocessors. It may also be provided using other technologies capable of performing logical operations such as, for example, AND, OR, and NOT, including but not limited to mechanical, optical, fluidic, and quantum technologies. In addition, the invention may be practiced within a general purpose computer or in any other circuits or systems as are known by those skilled in the art.

[065] As noted above, operand analysis circuit 12 analyzes the operand in the operand buffer 11 and generates signals providing information relating thereto. These signals are provided to the square root decision table logic circuit 15. In the exemplary embodiment, the operand analysis circuit 12 is implemented with a number of comparators, including:

[066] (i) a comparator 20 that generates an asserted signal if the bits $e_{msb}...e_{lsb}$ of the exponent field of the operand in buffer 11 are all binary one's, which will be the case if the operand is in the infinity format 75 or the NaN format 76;

[067] (ii) a comparator 21 that generates an asserted signal if the bits $e_{msb}...e_{lsb+1}$ of the exponent field of the operand in the buffer 11 are all binary one's and the bit e_{lsb} is a binary zero, which will be the case if the operand is in the overflow format 74;

[068] (iii) a comparator 22 that generates an asserted signal if the bit $e_{msb}...e_{lsb}$ of the exponent field of the operand in the buffer 11 are all binary zero's, which will be the case if the operand is in the zero format 70, underflow format 71, or denormalized format 72.

[069] (iv) a comparator 30 that generates an asserted signal if the bits $f_{msb}...f_{lsb+5}$ of the fraction field of the operand in the buffer 11 are all binary one's, which may be the case if the operand is in the denormalized format 72, normalized non-zero format 73, overflow format 74, or NaN format 76;

[070] (v) a comparator 31 that generates an asserted signal if the bits $f_{msb}...f_{lsb+5}$ of the fraction field of the operand in the buffer 11 are all binary zero's,

which may be the case if the operand is in the zero format 70, underflow format 72, denormalized format 72, normalized non-zero format 73 or infinity format 75;

[071] (vi) a comparator 32 that generates an asserted signal if the bits $f_{lsb+4} \dots f_{lsb}$ of the fraction field of the operand in the operand buffer 11 all binary one's, which may be the case if the operand is in the denormalized format 72 or normalized non-zero format 73, and which will be the case if the operand is in the overflow format 74, or if all of the flags "n", "o", "u", "z", and "x" are set in the infinity format 75 or NaN format 76;

[072] (vii) a comparator 33 that generates an asserted signal if the bits $f_{lsb+4} \dots f_{lsb+1}$ of the fraction field of the operand in the operand buffer 11 are all binary zero's, and if the bit f_{lsb} of the fraction field is either a binary "zero" or "one," which will be the case if the operand is in the zero format 70 or underflow format 71, and which may be the case if the operand is in the denormalized format 72, normalized non-zero format 73, overflow format 74, or if the flags "n", "o", "u", and "z" are clear and the flag "x" is either set or clear in the infinity format 75 or NaN format 76;

[073] (viii) a comparator 34 that generates an asserted signal if the bits $f_{lsb+4} \dots f_{lsb+1}$ of the fraction field of the operand in the buffer 11 are all binary zero's, and if the bit f_{lsb} of the fraction field is a binary "one," which will be the case if the operand is in the underflow format 71, and which may be the case if the operand is in the denormalized format 72, normalized non-zero format 73, or if the flags "n", "o", "u", and "z" are clear and the flag "x" is set in the infinity format 75 or NaN format 76; and

[074] (ix) a comparator 35 that generates an asserted signal if all of the bits $f_{lsb+4} \dots f_{lsb}$ of the fraction field of the operand in the buffer 11 are binary zero's, which will be the case if the operand is in the zero format 70, and which may be the case if the operand is in the denormalized format 72, normalized non-zero format 73, or if the flags "n", "o", "u", "z", and "x" are clear in the infinity format 75 or NaN format 76.

[075] In the illustrated embodiment, exemplary operand analysis circuit 12 also includes combinatorial logic elements that receive selected ones of the signals from the comparators and generates characteristic signals to provide indications as to certain characteristics of the respective operand. In more detail, such combinatorial logic elements include:

[076] (x) an AND gate 50, which will generate an asserted signal if comparators 31 and 35 are both generating asserted signals, which will be the case if the bits $f_{msb} \dots f_{lsb}$ of the fraction field of the operand in the operand buffer 11 have the bit pattern 000000000000000000000000;

[077] (xi) an AND gate 51, which will generate an asserted signal if comparators 31 and 34 are both generating asserted signals, which will be the case if the bits $f_{msb} \dots f_{lsb}$ of the fraction field of the operand in the operand buffer 11 have the bit pattern 000000000000000000000001;

[078] (xii) an AND gate 52, which will generate an asserted signal if comparators 30 and 32 are both generating asserted signals, which will be the case if the bits $f_{msb} \dots f_{lsb}$ of the fraction field of the operand in the operand buffer 11 have the bit pattern 111111111111111111111111;

[079] (xiii) an AND gate 40 that generates an asserted signal if the signals generated by both comparator 31 and comparator 33 are asserted, which will be the case if the operand is in the zero format 70 or underflow format 71 and which may be the case if the operand is in the normalized non-zero format 73, or if the flags "n," "o," "u," and "z" are clear and the flag "x" is either set or clear in the infinity format 75; otherwise stated, AND gate 40 will generate an asserted signal if the bits $f_{msb}...f_{lsb}$ of the fraction field of the operand in the operand buffer 11 has the bit pattern 000000000000000000000001 or 000000000000000000000000;

[080] (xiv) a NAND gate 45 that generates an asserted signal if the signal generated by comparator 22 is asserted and the signal generated by AND gate 40 is negated, which will be the case if the respective operand is in the denormalized format 72; otherwise stated, NAND gate 45 will generate an asserted signal if the bits $e_{msb}...e_{lsb}$ of the exponent field of the operand in the operand buffer 11 have the pattern 00000000 and a bit of the fraction field, other than the low order bit f_{lsb} , is a "one;"

[081] (xv) a NAND gate 46 that generates an asserted signal if all of comparator 20, comparator 21 or comparator 22 are generating negated signals, which will be the case if the bits $e_{msb}...e_{lsb}$ of the exponent field are not all "one's" or "zero's," and at least one bit, in addition to or other than the least significant bit e_{lsb} is also a "zero;"

[082] (xvi) a NAND gate 47 that generates an asserted signal if comparator 21 is generating an asserted signal and AND gate 52 is generating a negated signal, which will be the case if the bits $e_{msb}...e_{lsb}$ of the exponent field of the operand in the

respective operand buffer 11 have the bit pattern 11111110 and not all bits $f_{msb}...f_{lsb}$ of the fraction field of the operand in the respective operand buffer 11 are "one," so that the NAND gate 47 will generate an asserted signal if the bits $e_{msb}...e_{lsb}$ of the exponent field of the operand in the operand buffer 11 has the bit pattern 11111110 and at least one of the bits $f_{msb}...f_{lsb}$ of the fraction field of the operand in the same operand buffer 11 is "zero;" and

[083] (xvii) an OR gate 48 that generates an asserted signal if any of NAND gate 45, NAND gate 46 or NAND gate 47 is asserted, which will be the case if the operand in operand buffer 11 is in the denormalized format 72 or the normalized non-zero format 73.

[084] Each exemplary operand analysis circuit 12 provides signals to the square root decision table logic 15 as shown in the following table:

- (a) the sign bit from operand buffer 11;
- (b) the signal generated by comparator 22;
- (c) the signal generated by the comparator 21;
- (d) the signal generated by the comparator 20;
- (e) the signal generated by comparator 31;
- (f) the signal generated by AND gate 50;
- (g) the signal generated by AND gate 51;
- (h) the signal generated by AND gate 52;
- (i) the signal generated by the OR gate 48.

[085] In addition, signals from rounding mode store 16 representative of the rounding mode are provided to the square root decision table logic 15. In the illustrated embodiment, the square root decision table logic 15 generates control signals for controlling the result assembler 14. In addition, control signals generated by the square root decision table logic 15 control a set of XOR gates 41 and 42 that control toggling of one or both of the two least significant bits f_{lsb}, f_{lsb+1} of the fraction

field of the operand before it is provided to the square root core 13. As noted above, the result assembler 14 receives information from a number of sources, including the operand buffer 11, square root core 13 and several predetermined value stores as described below. Under control of control signals from the square root decision table logic circuit 15, the result assembler 14 assembles the appropriate result representing a square root of the operand onto a result bus 17. In general, the result assembler 14 essentially assembles the result in four segments, including a sign segment that represents the sign bit of the result, an exponent segment that represents the exponent field of the result, a high-order fraction segment that represents the bits $f_{msb} \dots f_{lsb+5}$ of the fraction field of the result, and a low-order fraction segment that represents the five least significant bits $f_{lsb+4} \dots f_{lsb}$ of the result. It will be appreciated that the low-order fraction segment in results in the infinity format 75 and NaN format 76 corresponds to the flags "n", "o", "u", "z", and "x." One or more of these segments will represent an embedded resulting status of the resulting floating point operand.

[086] In the illustrated embodiment, the result assembler includes four elements, including a link 60 carrying a control signal from the square root decision table logic 15, an exponent field selector 61, a high-order fraction field selector 62 and low-order fraction field combiner 63. The link 60 provides the sign of the result. Except in the case of an operand in the zero format 70 with a negative sign (that is, an operand "-0") the sign of the result is positive, in which case the sign bit would have the value "zero."

[087] As noted above, the exemplary selector 61 couples exponent value signals representative of the exponent field of the result to the result bus 17. In this embodiment, the selector 61 receives four sets of exponent field value signals, namely, the signals from the square root core 13 associated with the exponent field, as well as three sets of signals representative of three predetermined exponent field bit patterns, such as the patterns depicted in FIG. 1. It will be appreciated that these predetermined exponent field bit patterns correspond to the exponent fields associated with the zero format 70, underflow format 71, overflow format 74, infinity format 75 and NaN format 76. In addition, the selector 61 receives four exponent field control signals from the square root decision table logic 15. One of these control signals is associated with the signals from the square root core 13 that are associated with the exponent field. The rest of these control signals are associated with each of the sets of exponent field value signals, respectively. In enabling the result assembler 14 to assemble the result, the square root decision table logic 15 will assert one of the four exponent field control signals. Additionally, the selector 61 will couple the set of exponent field value signals associated with the asserted exponent field control signal to the result bus 17 to provide the exponent field of the result.

[088] The exemplary selector 62 couples high-order fraction field signals representative of the high-order fraction field bits $f_{msb} \dots f_{lsb+5}$ of the fraction field of the result to the result bus 17. In the illustrated embodiment, the selector 62 receives five sets of high-order fraction field value signals, namely, the signals from the square root core 13 associated with the high-order fraction field, signals

representative of bits $f_{msb} \dots f_{lsb+5}$ of the fraction field of the operand in operand buffer 11, as well as three sets of signals representative of three predetermined high-order fraction field bit patterns, such as the patterns depicted in FIG. 1. It will be appreciated that these predetermined high-order fraction field bit patterns correspond to the high-order fraction fields associated with the zero format 70, underflow format 71, overflow format 74, and infinity format 75, as well as the result (100000000000001100) in the NaN format 76 in which the information is encoded in the five bits $f_{lsb+9} \dots f_{lsb+5}$ of the fraction field indicating that the operand is less than zero. In addition, the selector 62 receives five high-order fraction field control signals from the square root decision table logic 15. One of these control signals is associated with the square root core 13. Another control signal is associated with the signals from operand buffer 11. The rest of these control signals are associated with each of the sets of high-order fraction field value signals, respectively. It will be appreciated that the control signal associated with the square root core 13 is the same control signal that controls the selector 61. In enabling the result assembler 14 to assemble the result, the square root decision table logic will assert one of the five high-order fraction field control signals, and the selector 62 will couple the set of high-order fraction field value signals associated with the asserted high-order fraction field control signal to the result bus 17 to provide bits $f_{msb} \dots f_{lsb+5}$ of the fraction field of the result.

[089] Similarly, the exemplary combiner 63 couples low-order fraction field value signals representative of the low-order fraction field bits $f_{lsb+4} \dots f_{lsb}$ of the fraction field of the result to the result bus 17. The combiner 63 receives three sets

of low-order fraction field signals, namely, the signals from the square root core 13 associated with the low-order fraction field, signals representative of bits $f_{lsb+4} \dots f_{lsb}$ of the fraction field of the operand in buffer 11, and one set of signals from the square root decision table logic 15. It will be appreciated that in the illustrated embodiment, the set of signals provided by the square root decision table logic 15 are used in controlling the condition of flags "n", "o", "u", "z", and "x" for those formats in which the low order bits $f_{lsb+4} \dots f_{lsb}$ represent flags. In addition, the sets of signals provided by the operand in buffer 11 may also represent the flags "n", "o", "u", "z", and "x." In addition, the combiner 63 receives two low-order fraction field control signals from the square root decision table logic 15. One control signal is associated with the set of low-order fraction field value signals provided by the square root core 13 and another is associated with the sets of signals provided by the operand buffer 11.

[090] In enabling the result assembler 14 to assemble the result, the square root decision table logic 15 may provide signals representative of the low-order fraction field and negate all of the low-order fraction field control signals. When this occurs, the signals representative of the low order fraction field provided by the square root decision table logic 15 will be coupled to the result bus 17 to provide bits $f_{lsb+4} \dots f_{lsb}$ of the fraction field of the result.

[091] Alternatively, the square root decision table logic 15 may negate all of the low-order fraction field value signals provided thereby and assert one of the two low-order fraction field control signals. When this occurs the combiner 63 will couple the set of low-order fraction field value signals associated with the asserted low-order fraction field control signal to the result bus 17 to provide bits $f_{lsb+4} \dots f_{lsb}$ of the

fraction field of the result. As a further alternative, the square root decision table logic 15 may negate all of the low-order fraction field value signals provided thereby and assert both of the two low-order fraction field control signals. As a result, the combiner 63 will couple the bit-wise OR of the sets of low-order fraction field value signals associated with the asserted low-order fraction field control signals to the result bus 17 to provide bits $f_{lsb+4} \dots f_{lsb}$ of the fraction field of the result. As yet another alternative, the square root decision table logic 15 may assert one or more of the low-order fraction field value signals provided thereby and assert one or more of the two low-order fraction field control signals. As a result, the combiner 63 will couple the bit-wise OR of the sets of low-order fraction field value signals associated with the asserted low-order fraction field control signals and the low-order fraction field value signals provided by the square root decision table logic 15 to the result bus 17 to provide bits $f_{lsb+4} \dots f_{lsb}$ the fraction field of the result.

[092] In more detail, the exemplary combiner 63 in the illustrated embodiment comprises an OR circuit 64 and two AND circuits 65 and 66. Each gate in the diagram may represent five gates. The AND circuits 65-66 receive the low-order fraction field value signals from the square root core 13 and operand buffer 11, respectively, as well as the respective low-order fraction field control signals. These AND circuits 65-66 perform a bit-wise AND operation to, if the respective low-order fraction field control signal is asserted, couple the low-order fraction field value signals to a respective input of OR circuit 64. The OR circuit 64, whose output is connected to the result bus 17, performs a bit-wise OR operation in connection with the signals that it receives from the AND circuits 65-66 and the low-order fraction

field value signals provided by the square root decision table logic 15. If the square root decision table logic 15 negates all of the low-order fraction field control signals, the AND circuits 65-66 will block the low-order fraction field value signals that they receive, and the signals provided by the OR circuit 64 will conform to the low-order fraction field value signals provided by the square root decision table logic 15.

[093] On the other hand, if the square root decision table logic 15 asserts one or more of the low-order fraction field control signals, the AND circuits 65-66 that receive the asserted low-order fraction field control signal will couple the low-order fraction field value signals that they receive to the OR circuit 64 and the other AND gates will block the low-order fraction field signal that they receive.

[094] As noted above, the square root decision table logic 15 generates the signal for link 60 and the control signals for controlling the selectors 61 and 62 and combiner 63 comprising the result assembler 14 and for controlling the toggling of the signals representing the low-order bits f_{lsb+1} and f_{lsb} from the operand buffer 11 before they are presented to the square root core 13. The control signals generated by the square root decision table logic 15 are such as to enable the result to be assembled in the desired format 70-76. Before proceeding further, it would be helpful to describe the results that are to be generated by the square root unit 10. Generally, results generated by the square root unit 10 are described in the table depicted in FIG. 3. In that table, "+P" means any finite positive nonzero representable value other than +UN (that is, a value in the underflow format 71, with the sign bit "s" being "zero") and +OV (that is, a value in the overflow format 74 with the sign bit "s" being "zero"); "-P" means any finite negative nonzero representable

value other than -UN (that is, a value in the underflow format 71, with the sign bit "s" being "one") and -OV (that is, a value in the overflow format 74, with the sign bit being "one"); and "NaN means any value whose exponent field is 11111111, other than one of the values represented by $+\infty$ (that is, a value in the infinity format 75, with the sign bit "s" being "zero") and $-\infty$ (that is, a value in the infinity format 75, with the sign bit "s" being "one").

[095] As noted above, square root decision table logic 15 generates the signal for link 60 and control signals for controlling the selectors 61, 62 and combiner 63 comprising the result assembler 14 and for controlling the toggling of the signals representing the low-order bits f_{lsb+1} and f_{lsb} from the operand buffer 11 before they are presented to the square root core 13. The particular signals that the square root decision table logic 15 will generate depends on the signals provided thereto by the operand buffer 11 representing the state of the operand sign bit, the operand analysis circuit 12, and rounding mode store 16. The series of input signals received by the square root decision table logic 15 are as follows:

[096] (a) The sign bit from operand buffer 11.

[097] (b) A signal from comparator 22 that is asserted if the exponent field of the operand in operand buffer 11 has the bit pattern 00000000;

[098] (c) A signal from comparator 21 that is asserted if the exponent field of the operand in operand buffer 11 has the bit pattern 11111110;

[099] (d) A signal from comparator 20 that is asserted if the exponent field of the operand in operand buffer 11 has the bit pattern 11111111;

[0100] (e) A signal from the comparator 31 that is asserted if the operand in operand buffer 11 has a high order fraction field with all 0-bits;

[0101] (f) A signal from AND gate 50 that is asserted if the operand in operand buffer 11 has high- and low-order fraction fields with the collective bit pattern 000000000000000000000000;

[0102] (g) A signal from AND gate 51 that is asserted if the operand in operand buffer 11 has high- and low-order fraction fields with the collective bit pattern 000000000000000000000001;

[0103] (h) A signal from AND gate 52 that is asserted if the operand in operand buffer 11 has high- and low-order fraction fields with the collective bit pattern 111111111111111111111111;

[0104] (i) A signal from OR gate 48 that is asserted if the operand in operand buffer 11 is in the denormalized format 72 or the normalized non-zero format 73;

[0105] (j) a signal from the rounding mode store 16 that is asserted if the rounding mode is either "round toward plus infinity" or "round toward minus infinity"; and

[0106] (k) a signal from the rounding mode store 16 that is asserted if the rounding mode is either "round toward zero" or "round toward minus infinity";

[0107] In response to these signals, the square root unit decision logic table 15 generates the following:

[0108] (1) a control signal that, if asserted, enables the second least-significant bit of the operand in operand buffer 11 to be toggled before presented to the square root core 13;

[0109] (2) a control signal that, if asserted, enables the least significant bit of the operand in operand buffer 11 to be toggled before presented to the square root core 13;

[0110] (3) a control signal that, if asserted, enables the exponent field and the high part of the fraction of the result to be provided by the remainder core 13, moreover, the five least-significant bits $f_{lsb+4} \dots f_{lsb}$ of the fraction field of the output provided by the remainder core 13 will contribute to the five least significant bits $f_{lsb+4} \dots f_{lsb}$ of the result;

[0111] (4) a control signal that corresponds to the sign bit of the result;

[0112] (5) a control signal that, if asserted, will enable the exponent field of the result to have the bit pattern 00000000:

[0113] (6) a control signal that, if asserted, will enable the exponent field of the result to have the bit pattern 11111110;

[0114] (7) a control signal that, if asserted, will enable the exponent field of the result to have the bit pattern 11111111;

[0115] (8) a control signal that, if asserted, will enable the high-order fraction of the result to correspond to the high-order fraction of the operand in operand buffer 11;

[0116] (9) a control signal that, if asserted, will enable the high-order fraction of the result to correspond to the bit pattern 00000000000000000000;

[0117] (10) a control signal that, if asserted, will enable the high-order fraction of the result to correspond to the bit pattern 1111111111111111;

[0118] (11) a control signal that, if asserted, will enable the high-order fraction of the result to correspond to the bit pattern 10000000000001100 (to represent a NaN value "square root of an operand that is less than zero").

[0119] (12) a control signal that, if asserted, will enable the low-order fraction field of the operand in output buffer 11 to contribute to the five least-significant bits $f_{lsb+4} \dots f_{lsb}$ of the fraction field of the result;

[0120] (13)-(17) signals that always contribute to the five least-significant bits $f_{lsb+4} \dots f_{lsb}$ of the fraction field of the result.

[0121] The specific patterns of output signals (1) through (17) generated by the exemplary square root decision table logic 15 in response to patterns of input signals (a) through (k) are depicted in FIG. 4. Generally, in FIG. 4, each row represents conditions of the output signals (1) through (17) that are generated by the square root decision table logic 15 in response to one pattern of input signals (a) through (k). In each row, the indicia to the left of the asterisk (*) represent the pattern of input signals (a) through (k) and the indicia to the right of the asterisk represent the pattern of output signals (1) through (17) with a "1" indicating that the respective input or output signal is asserted, a "0" indicating that the respective input or output signal is negated, and a "-" indicating that the respective input signal may be either negated or asserted. Each row is further annotated with an indication as to the respective format 70 through 76 of the operand in the operand buffer 11 and the format of the result.

[0122] Referring now to FIG. 4, a discussion of the first rows of input signal values and corresponding output signal values follows.

[0123] (A) for the two input patterns to the left of the asterisk:

[0124] (i) the first pattern "1--10----" indicates that signals (a) and (d) are asserted, signal (e) is negated, and signals (b), (c), and (f) through (i) may be either asserted or negated, with the pattern indicating a value in the NaN format 76 with a negative sign ("[-NaN]"); and

[0125] (ii) the second pattern "--" indicates that input signals (j) through (k), relating to the rounding mode, may be either asserted or negated; and

[0126] (B) for the seven output patterns to the right of the asterisk:

[0127] (i) the pattern "00" to the immediate right of the asterisk indicates that the signals provided to XOR gates 41 and 42, which control the toggling of the low-order bits f_{lsb+1} and f_{lsb} of the fraction fields of the operands in operand buffers 11A and 11B before being presented to square root core 13, are all negated;

[0128] (ii) the next "0" indicates that the signal provided to selectors 61 and 62 and AND circuit 65, is negated thereby to ensure that the output from square root core 13 will not contribute the result;

[0129] (iii) the next "0" indicates that the signal provided over link 60 is negated, so that the sign of the result will be positive;

[0130] (iv) the next pattern "001" indicates selector 61 is enabled to couple signals representative of the pattern 11111111 to the result bus 17, and the signals associated with the other patterns 00000000 and 11111110 will be negated;

[0131] (v) the next pattern "1000" indicates that the signal will be asserted that will enable selector 62 to couple the signals associated with the bits $f_{msb}...f_{lsb+5}$ comprising the high-order fraction field of the operand in buffer 11 to be coupled to the result bus 17 as the bits $f_{msb}...f_{lsb+5}$ of the fraction field of the result;

[0132] (vi) the next pattern "1" indicates that the signals provided by the square root decision table logic 15 will provide an asserted signal to AND circuits 66, enabling the AND gate to couple signals received thereby from operand buffers 11 to the OR circuit 64; and

[0133] (vii) the last pattern "00000" indicates that the signals provided by the square root decision table logic 15 to the OR circuit 64 are all negated. With such a last pattern, the OR gate will perform a bit-wise OR operation in connection with those signals and the signals provided thereto by AND circuits 65 and 66. Further, the negated signal described in (B)(ii) provides that the signals provided by AND circuit 65 are also negated, in which case the signals coupled by OR circuit 64 to result bus 17 will correspond to the bits $f_{lsb+4}...f_{lsb}$ from the fraction field of the operand in operand buffer 11.

[0134] On the right hand side of the first row in FIG. 4, the legend "[+NaN op1 f1]" indicates that the result value is in the NaN format 76, with a positive value, with the bits $f_{msb}...f_{lsb+5}$ of the fraction field of the result corresponding to bits $f_{msb}...f_{lsb+5}$ of the fraction field of the operand in operand buffer 11 and the bits $f_{lsb+4}...f_{lsb}$ of the result corresponding to the bits $f_{lsb+4}...f_{lsb}$ of the fraction fields of the operand in operand buffer 11. It should be noted that this corresponds to the result represented by the first line in the table depicted on FIG. 3.

[0135] In the context of the above discussion, the other rows will be apparent to those skilled in the art.

[0136] Square root decision table logic 15 may be implemented by many different circuit elements that will be apparent to those skilled in the art, including, but not limited to programmable logic arrays, ASIC circuits, general memory registers, other addressable memory storage devices, or a combination thereof.

[0137] One of ordinary skill in the art will recognize that other formats and bit patterns could be used to represent the floating point operand formats without departing from the principles of the present invention. One of ordinary skill in the art will also recognize that the floating point status information contained in the operands could easily be represented by other bit combinations (not shown) without departing from the principles of the present invention. For example, more or fewer bits could be used, a subset or superset of the exemplary status bits could be used, or the most significant bits of an operand (or some other subset of bits) could be used to indicate the floating point status information, instead of the least significant bits illustrated.

[0138] It will be appreciated that a system in accordance with an embodiment of the invention can be constructed in whole or in part from special purpose hardware or a general purpose computer system, or any combination thereof. Any portion of such a system may be controlled by a suitable program. Any program may in whole or in part comprise part of or be stored on the system in a conventional manner, or it may in whole or in part be provided in to the system over a network or other mechanism for transferring information in a conventional manner. In addition,

it will be appreciated that the system may be operated and/or otherwise controlled by means of information provided by an operator using operator input elements (not shown) which may be connected directly to the system or which may transfer the information to the system over a network or other mechanism for transferring information in a conventional manner.

[0139] The foregoing description has been limited to a specific embodiment of this invention. It will be apparent, however, that various variations and modifications may be made to the invention, with the attainment of some or all of the advantages of the invention. It is the object of the appended claims to cover these and such other variations and modifications as come within the true spirit and scope of the invention.

[0140] Other embodiments of the invention will be apparent to those skilled in the art from consideration of the specification and practice of the invention disclosed herein. It is intended that the specification and examples be considered as exemplary only, with a true scope and spirit of the invention being indicated by the following claims.